프로그래밍 언어론1

언어의 평가기준

1. 가독성 2. 작성력 3. 신뢰성 4. cost 5. 이식성 6. 일반성 7. 분명성

가독성	작성력	신뢰성	비용
- 단순성	- 단순성	- 타입체크	- 교육 - 컴파일(구현)
- 직교성	- 직교성	- 예외처리	- 신뢰성 - 실행
- 자료형	- 표현력	- Aliasing	- 코딩 - 유지보수
- 문법구조		- 가독성 & 작성력	

언어 디자인에 영향 끼친 요소

1. 컴퓨터 아키텍처 2. 프로그램 디자인 방법론

1. 폰 노이먼 구조

CPU ↔ main memory (Data and Programs) Fetch-execute-cycle

⇓

Imperative language 특징 Bottle neck

- 변수 - 할당문 - Iteration

2.

1950~1960	machine efficiency
1960	People efficiency
1970	Process-oriented → Data-oriented
1980	Object-oriented

언어 종류

1. 명령형 2. 함수형 3. [illegible]

1. 명령 2. 함수형 3. 논리 언어 4. Mark up

Trade - off

Implementation Methods

1. 컴파일 2. Pure 3. Hybrid

Syntax 문법 semantics 의미

lexeme 최소의미단어 Token 렉셈의 카테고리

BNF

: Context-free grammars

LHS, RHS

<assign> → <var> = <expr>

<assign> : non terminal

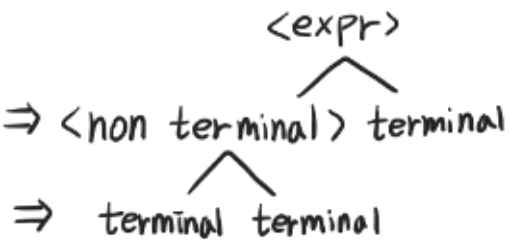
= : terminal

<id> → A|B|C

<expr> → <id>|<id> + <expr>

derivation (⇒)

Parse Tree



remove
Unambiguous factor !!

1. precedence level
2. associativity
3. selector

EBNF

[ ] : optional (생략가능)

( ) : Parentheses

{ } : 0 or more (없거나 반복)

AGS = CFG + Semantic info

Semantics methods

- operational : 변수의 상태 전이 과정으로 설명

- denotational : 수학적 함수를 구성하여 정의

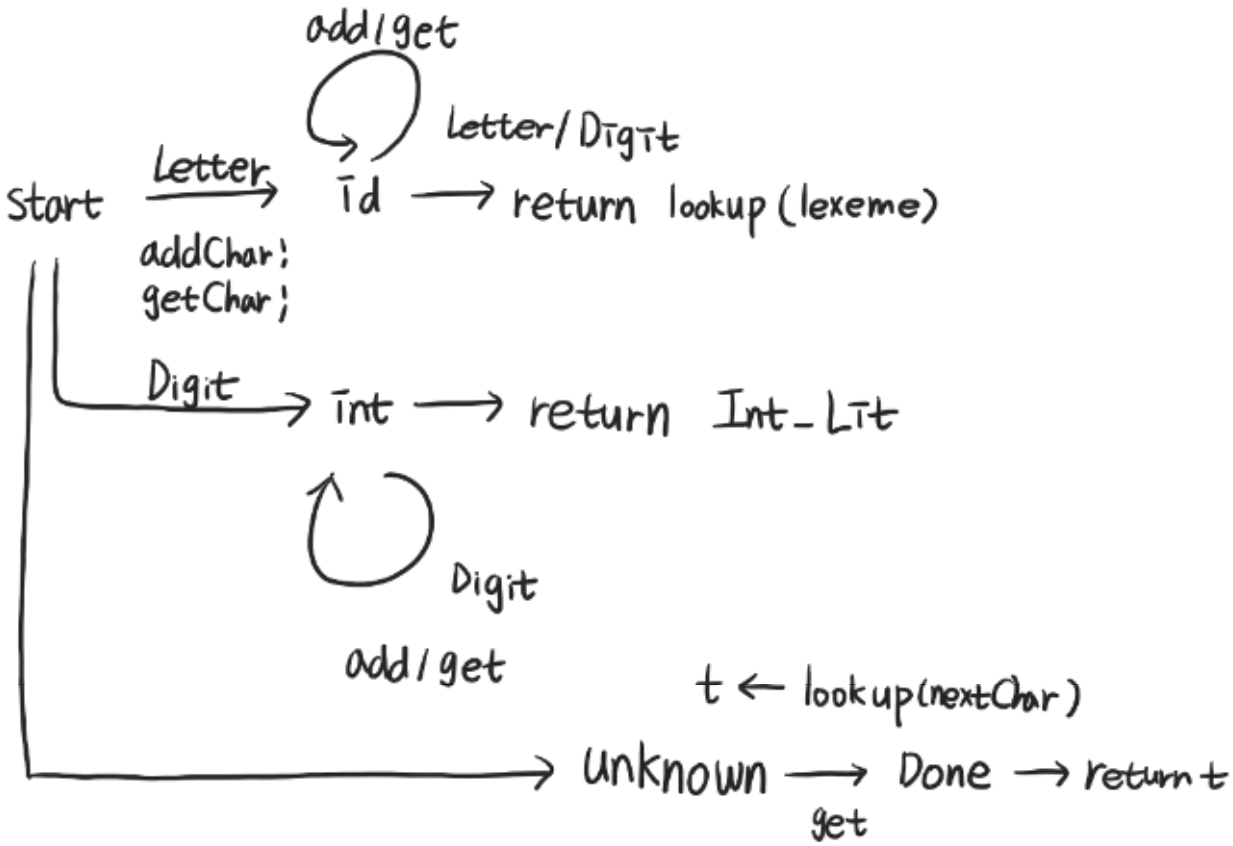
- axiomatic : 논리적 선언 형태로 정의

Syntax Analysis

1. lexical analyzer

   a pattern matcher

<state Diagram>



Top-Down Parser (leftmost derivation)

- Recursive descent (code implementation)

<expr> → <term> {(+|-) <term>}

⇒ expr(){

  term();

lex() : Puts the next token code in

```
while( nextToken +|- ){
    lex(); term();
  }
}
```

<term> → <factor> {(*|/)<factor>}

⇒

```
term(){
    factor();
    while (nextToken *|/ ){ lex(); factor();}
}
```

<factor> → id | int_constant | (<expr>)

⇒

```
factor(){
    if (nextToken id | int) {lex();}
    else if (nextToken "(" ){
        lex();
        expr();
        if ( next Token ")" ){ lex();}
        else { error();}
    }
    else{ error();}
}
```

Bottom - Up Parser

nextToken

ex) (sum + 47) / total

⇒ lex() 실행 nextChar = '('

enter expr(), term(), factor()

lex() 실행 nextChar = 'sum'

enter expr(), term(), factor()

lex() 실행 nextChar = '+'

exit factor(), term()

lex() 실행 nextChar = '47'

enter term(), factor()

lex() 실행 nextChar = ')'

exit factor(), term(), expr()

lex() 실행 nextChar = '/'

exit factor()

lex() 실행 nextChar = 'total'

enter factor()

lex() 실행 nextChar = '\0'

- Shift-reduce Parser

  Shift : 스택에 토큰을 Push

  reduce : handle을 replace

- LR Parser

  Stack : State+Token $

  Action table : (state, token)에 해당하는 액션을 한다.

  GoTo table : reduce 작업시 replace된 Token에 State를 추가한다.

stack	input	Action
0	id+id*id$	Shift 5 — new state
0id5	+id*id$	Reduce 6 (Goto[0,F]) — state, new Token
0F3 T→F	+id*id$	Reduce 4 (Goto[0,T])
0T2 F→T	+id*id$	Reduce 2 (Goto[0,E])
0E1 T→E	+id*id$	Shift 6
0E1+6	id*id$	Shift 5
0E1+6id5	*id$	Reduce 6 (Goto[6,F])
0E1+6F3	*id$	Reduce 4 (Goto[6,T])
0E1+6T9	*id$	Shift 7
0E1+6T9*7	id$	Shift 5
0E1+6T9*7id5	$	Reduce 6 (Goto[7,F])
0E1+6T9*7F10	$ T←TF	Reduce 3 (Goto[6,T])
0E1+6T9	$	Reduce 1 (Goto[0, E])
0E1	$	accept

Design Issues for name

- Length - special characters - Case sensitivity - special words

no limit PHP($) Perl($,@,%) C(O) others(X)

Ruby(@,@@)

Variable

: an abstraction of a memory cell

Attribute of attributes

- Name -Address - Value - Type - Life time -Scope

range/operation binding

binding time Ocurr duration

- language design time : operator operation
- language implementation time : float type representation
- Compile time : variable type
- Load time : Static memory
- Run time : local memory

(Static binding (run) : Implicit (Basic, Perl, Ruby, JS, PHP)
Dynamic binding (execute) : JS, Python, Ruby, PHP, C#

life time

Static : bound to memory before execution begins
- Stack-dynamic : declaration statements executed
- Explicit heap-dynamic : execution
- Implicit heap-dynamic

Scope (range of visible)

- Scheme

(Static Scope
Dynamic Scope (Call 상태)

```
(LET (
     상태 (top (+ a b))
```

```
        (bottom (- c d)))
        명령 (/ top bottom)
)
```

- ML

```
let
    선언 val top = a +  b
         val bottom = c - d
in
    명령 top / bottom
end;
```

- F#

```
let n1 =
    (let n2 = 7
     let n3 = n2 + 3
     n3);
let n4 = n3 + n1;;
```

- PHP

```
$day = "Monday";
$month = "January";

function calendar(){
        $day = "Tuesday"; // local
        global $month;  // Global
        print "local day is $day <br />";
        $gday = $GLOBALS['day']; // 글로벌변수들 배열
        print "global day is $gday <br />";
        print "global month is $month <br />";
}

calendar();
```

Global 선언시
그 변수 참조가능

```
local day is Tuesday
global day is Monday
global month is January
```

- Python

```
day = "Monday"
def tester():
        print "The global day is: ", day 읽기가능
tester()
```

```
The global day is : Monday
```

```
day = "Monday"
def tester():
        print "The global day is: ", day // local로 판단하고 에러 (day가 함수내 있기에)
        day = "Tuesday"
        print "The new value of day is: ", day
tester()
```

local

```
UnboundLocalError
```

```
day = "Monday"
def tester():
        global day   write 글로벌사용선언
        print "The global day is: ", day // Global
```

Global (write) 쓰기위해서
function 내 Global 선언

```
        day = "Tuesday" // Global 재선언
        print "The new value of day is: ", day
tester()
```

```
The global day is : Monday
The new value of day is : Tuesday
```